(19) Europäisch s Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 102 165 A1

# (12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
**23.05.2001 Bulletin 2001/21**

(51) Int Cl.[7]: **G06F 9/38**

(21) Application number: **00310114.4**

(22) Date of filing: **14.11.2000**

(84) Designated Contracting States:
**AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE TR**
Designated Extension States:
**AL LT LV MK RO SI**

(30) Priority: **15.11.1999 US 165512 P**
**18.02.2000 US 183527 P**
**18.02.2000 US 183609 P**

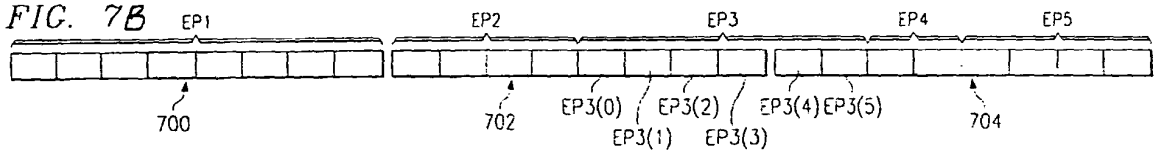
(71) Applicant: **Texas Instruments Incorporated**
**Dallas, Texas 75251 (US)**

(72) Inventors:
- **Simar, Jr. Laurence R.**
  **Texas 77469 (US)**
- **Simar, Laurence**
  **Houston Texas 77057 (US)**

(74) Representative:
**Legg, Cyrus James Grahame et al**
**ABEL & IMRAY,**
**20 Red Lion Street**
**London WC1R 4PQ (GB)**

## (54) Microprocessor with execution packet spanning two or more fetch packets

(57) A data processing system with a microprocessor (10). The microprocessor has in instruction execution pipeline includes fetch and decode stages and several functional execution units (L1/2, S1/2, M1/2, D1/2). Fetch packets (700, 702, 704) contain a plurality of instruction words. Execution packets (EP1 ... EP5) include a plurality of instruction words that can be executed in parallel by two or more execution units. An execution packet can span two or more fetch packets.




EP 1 102 165 A1

## Description

**[0001]** This invention relates to data processing devices, electronic processing and control systems and methods of their manufacture and operation, and particularly relates to memory access schemes of microprocessors optimized for digital signal processing.

**[0002]** Generally, a microprocessor is a circuit that combines the instruction-handling, arithmetic, and logical operations of a computer on a single semiconductor integrated circuit. Microprocessors can be grouped into two general classes, namely general-purpose microprocessors and special-purpose microprocessors. General-purpose microprocessors are designed to be programmable by the user to perform any of a wide range of tasks, and are therefore often used as the central processing unit (CPU) in equipment such as personal computers. Special-purpose microprocessors, in contrast, are designed to provide performance improvement for specific predetermined arithmetic and logical functions for which the user intends to use the microprocessor. By knowing the primary function of the microprocessor, the designer can structure the microprocessor architecture in such a manner that the performance of the specific function by the special-purpose microprocessor greatly exceeds the performance of the same function by a general-purpose microprocessor regardless of the program implemented by the user.

**[0003]** One such function that can be performed by a special-purpose microprocessor at a greatly improved rate is digital signal processing. Digital signal processing generally involves the representation, transmission, and manipulation of signals, using numerical techniques and a type of special-purpose microprocessor known as a digital signal processor (DSP). Digital signal processing typically requires the manipulation of large volumes of data, and a digital signal processor is optimized to efficiently perform the intensive computation and memory access operations associated with this data manipulation. For example, computations for performing Fast Fourier Transforms (FFTs) and for implementing digital filters consist to a large degree of repetitive operations such as multiply-and-add and multiple-bit-shift. DSPs can be specifically adapted for these repetitive functions, and provide a substantial performance improvement over general-purpose microprocessors in, for example, real-time applications such as image and speech processing.

**[0004]** DSPs are central to the operation of many of today's electronic products, such as high-speed modems, high-density disk drives, digital cellular phones, complex automotive systems, and video-conferencing equipment. DSPs will enable a wide variety of other digital systems in the future, such as video-phones, network processing, natural speech interfaces, and ultra-high speed modems. The demands placed upon DSPs in these and other applications continue to grow as consumers seek increased performance from their digital products, and as the convergence of the communications, computer and consumer industries creates completely new digital products.

**[0005]** Microprocessor designers have increasingly endeavored to exploit parallelism to improve performance. One parallel architecture that has found application in some modern microprocessors utilizes instruction fetch packets and multiple instruction execution packets with multiple functional units, referred to as a Very Long Instruction Word (VLIW) architecture.

**[0006]** Digital systems designed on a single integrated circuit are referred to as an application specific integrated circuit (ASIC). MegaModules are being used in the design of ASICs to create complex digital systems a single chip. (MegaModule is a trademark of Texas Instruments Incorporated.) Types of MegaModules include SRAMs, FIFOs, register files, RAMs, ROMs, universal asynchronous receiver-transmitters (UARTs), programmable logic arrays and other such logic circuits. MegaModules are usually defined as integrated circuit modules of at least 500 gates in complexity and having a complex ASIC macro function. These MegaModules are predesigned and stored in an ASIC design library. The MegaModules can then be selected by a designer and placed within a certain area on a new IC chip.

**[0007]** Designers have succeeded in increasing the performance of DSPs, and microprocessors in general, by increasing clock speeds, by removing data processing bottlenecks in circuit architecture, by incorporating multiple execution units on a single processor circuit, and by developing optimizing compilers that schedule operations to be executed by the processor in an efficient manner. However, execution packets in VLIW architectures must be aligned within a single instruction fetch packet.

**[0008]** The increasing demands of technology and the marketplace make desirable even further structural and process improvements in processing devices, application systems and methods of operation and manufacture.

**[0009]** The invention provides a microprocessor comprising:

fetch circuitry for fetching instruction fetch packets, wherein each fetch packet contains a first plurality of instructions;

a plurality of functional units operable to execute a second plurality of instructions in parallel, and

dispatch circuitry operable to select an execute packet from one or more fetch packets, wherein an execute packet varies in size and contains only a set of instructions that can be executed in parallel on the plurality of functional units, whereby a first execute packet contains a different number of instructions from a second execute packet due to resource constraints.

**[0010]** In one embodiment, the first plurality of instructions is equal in number to the second plurality of instructions. Preferably, a first execute packet spans a first fetch packet and a second fetch packet.

**[0011]** Preferably, the dispatch circuitry comprises:

a first latch to hold the first fetch packet;
a second latch to hold the second fetch packet;
selection circuitry to select a first portion of the first execute packet from the first latch and a second portion of the first execute packet from the second latch.

**[0012]** Preferably, the dispatch circuitry further comprises control circuitry connected to a plurality of instruction positions corresponding to the plurality of instructions of a fetch packet in the first latch and in the second latch to determine a boundary of each execute packet, the control circuitry operable to control the selection circuitry in response to an execute packet boundary.

**[0013]** Preferably, the microprocessor is included in a digital processing system.

**[0014]** Preferably, the digital processing system is a cellular telephone, further comprising:

an integrated keyboard connected to the microprocessor via a keyboard adapter;
a display, connected to the microprocessor via a display adapter;
radio frequency (RF) circuitry connected to the microprocessor; and an aerial connected to the RF circuitry.

**[0015]** The invention provides a method of operating a microprocessor having a plurality of functional units for executing instructions in parallel, comprising the steps of:

fetching a sequence of instruction fetch packets, wherein each fetch packet contains a first plurality of instructions;
examining each fetch packet to determine an execute packet boundary and;
selecting a first portion of an execute packet from a first fetch packet and a second portion of a first execute packet from a second fetch packet if the first execute packet boundaries span the first fetch packet and the second fetch packet.

**[0016]** The invention provides a method of operating a digital system including a microprocessor having a plurality of functional units for executing instructions in parallel, comprising the steps of:

fetching a sequence of instruction fetch packets, wherein each fetch packet contains a first plurality of instructions;
examining each fetch packet to determine an execute packet boundary and;
selecting a first portion of an execute packet from a first fetch packet and a second portion of a first execute packet from a second fetch packet if the first execute packet boundaries span the first fetch packet and the second fetch packet.

**[0017]** An illustrative embodiment of the present invention seeks to provide a microprocessor and a method for accessing memory by a microprocessor that improves digital signal processing performance. Aspects of the invention are specified in the claims.

**[0018]** In an embodiment of the present invention, fetch packets contain a plurality of instruction words. Execution packets include a plurality of instruction words that can be executed in parallel by two or more execution units. An execution packet can span two or more fetch packets.

**[0019]** In an embodiment of the present invention, there are no execute packet boundary restrictions, thereby eliminating a need to pad a fetch packet by adding unneeded NOP instructions.

**[0020]** Other features and advantages of the present invention will become apparent by reference to the following detailed description when considered in conjunction with the accompanying drawings, in which:

Figure 1 is a block diagram of a digital signal processor (DSP), showing components thereof pertinent to an embodiment of the present invention;
Figure 2 is a block diagram of the functional units, data paths and register files of Figure 1;
Figure 3 shows the basic format of a fetch packet of the DSP of Figure 1;
Figure 4A depicts a fetch packet of Figure 3 with fully serial p-bits;
Figure 4B depicts a fetch packet of Figure 3 with fully parallel p-bits;
Figure 4C depicts a fetch packet of Figure 3 with partially serial p-bits;
Figure 5 depicts fetch packet n, which contains three execute packets, shown followed by six fetch packets n+1 through n+6, each with one execution packet containing 8 parallel instructions;

Figure 6 is a block diagram of a prior art processor, illustrating how execution packets must be aligned within fetch packets;

Figure 7A is another illustration of fetch packets and execution packets in the prior art processor of Figure 6;

Figure 7B is an illustration of execution packets spanning fetch packets for the processor of Figure 1;

Figure 8 is a block diagram of the processor of Figure 1, illustrating a sequence of execution packets spanning fetch packets;

Figure 9 is a block diagram of a portion of the instruction fetch pipeline of the processor of Figure 1 that illustrates circuitry for dispatching an execution packet that spans two fetch packets;

Figure 10 is a block diagram of an alternative embodiment of the processor of Figure 1; and

Figure 11 is a wireless telephone that embodies the present invention.

**[0021]** Figure 1 is a block diagram of a microprocessor 1 which has an embodiment of the present invention. Microprocessor 1 is a VLIW digital signal processor ("DSP"). In the interest of clarity, Figure 1 only shows those portions of microprocessor 1 that are relevant to an understanding of an embodiment of the present invention. Details of general construction for DSPs are well known, and may be found readily elsewhere. For example, U.S. Patent 5,072,418 issued to Frederick Boutaud, et al, describes a DSP in detail and is incorporated herein by reference. U.S. Patent 5,329,471 issued to Gary Swoboda, et al, describes in detail how to test and emulate a DSP and is incorporated herein by reference. Details of portions of microprocessor 1 relevant to an embodiment of the present invention are explained in sufficient detail hereinbelow, so as to enable one of ordinary skill in the microprocessor art to make and use the invention.

**[0022]** In microprocessor 1 there are shown a central processing unit (CPU) 10, data memory 22, program memory 23, peripherals 60 and an external memory interface (EMIF) with a direct memory access (DMA) 61. CPU 10 further has an instruction fetch/decode unit 10a-c , a plurality of execution units, including an arithmetic and load/store unit D1, a multiplier M1, an ALU/shifter unit S1, an arithmetic logic unit ("ALU") L1, a shared multi-port register file 20a from which data are read and to which data are written. Decoded instructions are provided from the instruction fetch/decode unit 10a-c to the functional units D1, M1, S1, and L1 over various sets of control lines which are not shown. Data are provided to/from the register file 20a from/to to load/store units D1 over a first set of busses 32a, to multiplier M1 over a second set of busses 34a, to ALU/shifter unit S1 over a third set of busses 36a and to ALU L1 over a fourth set of busses 38a. Data are provided to/from the memory 22 from/to the load/store units D1 via a fifth set of busses 40a. Note that the entire data path described above is duplicated with register file 20b and execution units D2, M2, S2, and L2. Instructions are fetched by fetch unit 10a from instruction memory 23 over a set of busses 41. Emulation circuitry 50 provides access to the internal operation of integrated circuit 1 which can be controlled by an external test/development system (XDS) 51.

**[0023]** External test system 51 is representative of a variety of known test systems for debugging and emulating integrated circuits. One such system is described in U.S. Patent 5,535,331 which is incorporated herein by reference. Test circuitry 52 contains control registers and parallel signature analysis circuitry for testing integrated circuit 1.

**[0024]** Note that the memory 22 and memory 23 are shown in Figure 1 to be a part of a microprocessor 1 integrated circuit, the extent of which is represented by the box 42. The memories 22-23 could just as well be external to the microprocessor 1 integrated circuit 42, or part of it could reside on the integrated circuit 42 and part of it be external to the integrated circuit 42. These are matters of design choice. Also, the particular selection and number of execution units are a matter of design choice, and are not critical to the invention.

**[0025]** When microprocessor 1 is incorporated in a data processing system, additional memory or peripherals may be connected to microprocessor 1, as illustrated in Figure 1. For example, Random Access Memory (RAM) 70, a Read Only Memory (ROM) 71 and a Disk 72 are shown connected via an external bus 73. Bus 73 is connected to the External Memory Interface (EMIF) which is part of functional block 61 within microprocessor 42. A Direct Memory Access (DMA) controller is also included within block 61. The DMA controller is generally used to move data between memory and peripherals within microprocessor 1 and memory and peripherals which are external to microprocessor 1.

**[0026]** A detailed description of various architectural features of the microprocessor of Figure 1 is provided in coassigned application S.N. 09/012,813 (TI-25311) and is incorporated herein by reference.

**[0027]** Figure 2 is a block diagram of the execution units and register files of the microprocessor of Figure 1 and shows a more detailed view of the buses connecting the various functional blocks. In this figure, all data busses are 32 bits wide, unless otherwise noted. There are two general-purpose register files (A and B) in the processor's data paths. Each of these files contains 32 32-bit registers (A0-A31 for file A and B0-B31 for file B). The general-purpose registers can be used for data, data address pointers, or condition registers. Any number of reads of a given register can be performed in a given cycle.

**[0028]** The general-purpose register files support data ranging in size from packed 8-bit data through 64-bit fixed-point data. Values larger than 32 bits, such as 40-bit long and 64-bit double word quantities, are stored in register pairs, with the 32 LSBs of data placed in an even-numbered register and the remaining 8 or 32 MSBs in the next upper

register (which is always an odd-numbered register). Packed data types store either four 8-bit values or two 16-bit values in a single 32-bit register.

**[0029]** There are 32 valid register pairs for 40-bit and 64-bit data, as shown in Table 1. In assembly language syntax, a colon between the register names denotes the register pairs and the odd numbered register is specified first.

Table 1.

| 40-Bit/64-Bit Register Pairs | |
|---|---|
| Register Files | |
| A | B |
| A1:A0 | B1:B0 |
| A3:A2 | B3:B2 |
| A5:A4 | B5:B4 |
| A7:A6 | 67:B6 |
| A9:A8 | B9:B8 |
| A11:A10 | B11:B10 |
| A13:A12 | B13:B12 |
| A15:A14 | B15:B14 |
| A17:A16 | B17:B16 |
| A19:A18 | B19:B18 |
| A21:A20 | B21:B20 |
| A23:A22 | B23:B22 |
| A25:A24 | B25:B24 |
| A27:A26 | B27:B26 |
| A29:A28 | B29:B28 |
| A31:A30 | B31:B30 |

**[0030]** The eight functional units in processor 10's data paths can be divided into two groups of four; each functional unit in one data path is almost identical to the corresponding unit in the other data path. The functional units are described in Table 2.

**[0031]** Besides being able to perform 32-bit data manipulations, processor 10 also contains many 8-bit and 16-bit data instructions in the instruction set. For example, the MPYU4 instruction performs four 8x8 unsigned multiplies with a single instruction on an .M unit. The ADD4 instruction performs four 8-bit additions with a single instruction on an .L unit.

Table 2.

| Functional Units and Operations Performed | |
|---|---|
| Functional Unit<br>Operations | Fixed-Point |
| | |
| .L unit (.L1, .L2) arithmetic and compare operations | 32/40-bit |
| logical operations | 32-bit |
| Leftmost 1 or 0 counting for 32 bits | |
| Normalization count for 32 and 40 bits | |

Table 2. (continued)

| Functional Units and Operations Performed | |
|---|---|
| Functional Unit<br>Operations | Fixed-Point |
| shifts | Byte |
| packing/unpacking | Data |
| constant generation | 5-bit |
| 16-bit arithmetic operations<br>bit arithmetic operations | Paired<br>Quad 8- |
| 16-bit min/max operations<br>bit min/max operations | Paired<br>Quad 8- |
| | |
| .s unit (.s1, .S2) arithmetic operations | 32-bit |
| bit shifts and 32-bit bit-field operations | 32/40- |
| logical operations | 32-bit |
| Branches | |
| Constant generation | |
| Register transfers to/from control register file (.S2 only) | |
| shifts | Byte |
| packing/unpacking | Data |
| 16-bit compare operations<br>8-bit compare operations | Paired<br>Quad |
| 16-bit shift operations<br>16-bit saturated arithmetic operations<br>8-bit saturated arithmetic operations | Paired<br>Paired<br>Quad |
| | |
| .M unit (.M1, .M2)<br>operations | 16 x 16 multiply |

Table 2. (continued)

| Functional Units and Operations Performed | |
|---|---|
| Functional Unit<br>Operations | Fixed-Point |
| multiply operations | 16 x 32 |
| expansion | Bit |
| interleaving/de-interleaving | Bit |
| x 8 multiply operations | Quad 8 |
| 16 x 16 multiply operations | Paired |
| 16 x 16 multiply with add/subtract operations | Paired<br><br>Quad 8 |
| x 8 multiply with add operations | |
| Variable shift operations<br><br>Rotation<br><br>Field Multiply | <br><br>Galois |
| | |
| .D unit (.D1, .D2)<br>subtract, linear and circular address calculation | 32-bit add, |
| and stores with 5-bit constant offset | Loads |
| and stores with 15-bit constant offset (.D2 only) | Loads |
| and store double words with 5-bit constant | Load |
| and store non-aligned words and double words | Load |
| constant generation | 5-bit |
| bit logical operations | 32- |

**[0032]** Most data lines in the CPU support 32-bit operands, and some support long (40-bit) and double word (64-bit) operands. Each functional unit has its own 32-bit write port into a general-purpose register file (Refer to Figure 2A). All units ending in 1 (for example, .L1) write to register file A and all units ending in 2 write to register file B. Each functional unit has two 32-bit read ports for source operands src1 and src2. Four units (.L1, .L2, .S1, and .S2) have an extra 8-bit-wide port for 40-bit long writes, as well as an 8-bit input for 40-bit long reads. Because each unit has its own 32-bit write port, when performing 32-bit operations all eight units can be used in parallel every cycle. Since each multiplier can return up to a 64-bit result, two write ports are provided from the multipliers to the register file.

Register File Cross Paths

**[0033]** Each functional unit reads directly from and writes directly to the register file within its own data path. That is, the .L1, .S1, .D1, and .Ml units write to register file A and the .L2, .S2, .D2, and .M2 units write to register file B. The register files are connected to the opposite-side register file's functional units via the 1X and 2X cross paths. These cross paths allow functional units from one data path to access a 32-bit operand from the opposite side's register file. The 1X cross path allows data path A's functional units to read their source from register file B. Similarly, the 2X cross path allows data path B's functional units to read their source from register file A.

**[0034]** All eight of the functional units have access to the opposite side's register file via a cross path. The .M1, .M2, .S1, .S2, D1 and .D2 units' src2 inputs are selectable between the cross path and the same side register file. In the case of the .L1 and .L2 both srcl and src2 inputs are also selectable between the cross path and the same-side register file.

**[0035]** Only two cross paths, 1X and 2X, exist in this embodiment of the architecture. Thus the limit is one source read from each data path's opposite register file per cycle, or a total of two cross-path source reads per cycle. Advantageously, multiple units on a side may read the same cross-path source simultaneously. Thus the cross path operand for one side may be used by any one, multiple or all the functional units on that side in an execute packet. In the C62x/C67x, only one functional unit per data path, per execute packet could get an operand from the opposite register file.

**[0036]** A delay clock cycle is introduced whenever an instruction attempts to read a register via a cross path that was updated in the previous cycle. This is known as a cross path stall. This stall is inserted automatically by the hardware; no NOP instruction is needed. It should be noted that no stall is introduced if the register being read is the destination for data loaded by a LDx instruction.

Memory, Load and Store Paths

**[0037]** Processor 10 supports double word loads and stores. There are four 32-bit paths for loading data for memory to the register file. For side A, LD1a is the load path for the 32 LSBs; LD1b is the load path for the 32 MSBs. For side B, LD2a is the load path for the 32 LSBs; LD2b is the load path for the 32 MSBs. There are also four 32-bit paths, for storing register values to memory from each register file. ST1a is the write path for the 32 LSBs on side A; ST1b is the write path for the 32 MSBs for side A. For side B, ST2a is the write path for the 32 LSBs; ST2b is the write path for the 32 MSBs.

**[0038]** Some of the ports for long and double word operands are shared between functional units. This places a constraint on which long or double word operations can be scheduled on a datapath in the same execute packet.

Data Address Paths

**[0039]** Bus 40a has an address bus DA1 which is driven by mux 200a. This allows an address generated by either load/store unit D1 or D2 to provide a memory address for loads or stores for register file 20a. Data Bus LD1 loads data from an address in memory 22 specified by address bus DA1 to a register in load unit D1. Unit D1 may manipulate the data provided prior to storing it in register file 20a. Likewise, data bus ST1 stores data from register file 20a to memory 22. Load/store unit D1 performs the following operations: 32-bit add, subtract, linear and circular address calculations. Load/store unit D2 operates similarly to unit D1, with the assistance of mux 200b for selecting an address.

**[0040]** The DA1 and DA2 resources and their associated data paths are specified as T1 and T2 respectively. T1 consists of the DA1 address path and the LD1a, LD1b, ST1a and ST1b data paths. Similarly, T2 consists of the DA2 address path and the LD2a, LD2b, ST2a and ST2b data paths. The T1 and T2 designations appear in functional unit fields for load and store instructions.

**[0041]** For example, the following load instruction uses the .D1 unit to generate the address but is using the LD2a path resource from DA2 to place the data in the B register file. The use of the DA2 resource is indicated with the T2 designation.

```
LDW     .D1T2     *A0[3], B1
```

**[0042]** Table 3 defines the mapping between instructions and functional units for a set of basic instructions included in DSP 10 is described in U.S. Patent S.N. 09/012,813 (TI-25311, incorporated herein by reference). Table 4 defines a mapping between instructions and functional units for a set of extended instructions in an embodiment of the present invention. A complete description of the extended instructions is provided in U.S. Patent S.N. _____ (TI-30302) entitled "Microprocessor with Improved ISA," and is incorporated herein by reference. Alternative embodiments of the present invention may have different sets of instructions and functional unit mapping. Table 3 and Table 4 are illustrative and are not exhaustive or intended to limit various embodiments of the present invention.

Table 3.

| Instruction to Functional Unit Mapping of Basic Instructions | | | |
|---|---|---|---|
| .L Unit | .M Unit | .S Unit | .D Unit |
| ABS | MPY | ADD | ADD |
| ADD | SMPY | ADDK | ADDA |
| AND | | ADD2 | LD mem |
| CMPEQ | | AND | LD mem (15-bit offset) (D2 only) |
| CMPGT | | B disp | MV |
| CMPGTU | | B IRP | NEG |
| CMPLT | | B NRP | ST mem |
| CMPLTU | | B reg | ST mem (15-bit offset) (D2 only) |
| LMBD | | CLR | SUB |
| MV | | EXT | SUBA |
| NEG | | EXTU | ZERO |
| NORM | | MVC | |
| NOT | | MV | |
| OR | | MVK | |
| SADD | | MVKH | |
| SAT | | NEG | |
| SSUB | | NOT | |
| SUB | | OR | |
| SUBC | | SET | |
| XOR | | SHL | |
| ZERO | | SHR | |
| | | SHRU | |
| | | SSHL | |
| | | STP (S2 only) | |
| | | SUB | |
| | | SUB2 | |
| | | XOR | |
| | | ZERO | |

Table 4.

| Instruction to Functional Unit Mapping of Extended Instructions | | | |
|---|---|---|---|
| .L unit | .M unit | .S unit | .D unit |
| ABS2 | AVG2 | ADD2 | ADD2 |
| ADD2 | AVGU4 | ADDKPC | AND |
| ADD4 | BITC4 | AND | ANDN |
| AND | BITR | ANDN | LDDW |

Table 4. (continued)

| Instruction to Functional Unit Mapping of Extended Instructions | | | |
|---|---|---|---|
| .L unit | .M unit | .S unit | .D unit |
| ANDN | DEAL | BDEC | LDNDW |
| MAX2 | DOTP2 | BNOP | LDNW |
| MAXU4 | DOTPN2 | BPOS | MVK |
| MIN2 | DOTPNRSU2 | CMPEQ2 | OR |
| MINU4 | DOTPNRUS2<br>DOTPRSU2<br>DOTPRUS2 | CMPEQ4<br>CMPGT2<br>CMPGTU4 | STDW |
| MVK | DOTPSU4<br>DOTPUS4 | CMPLT2 | STNDW |
| OR | DOTPU4 | CMPLTU4 | STNW |
| PACK2 | GMPY4 | MVK | SUB2 |
| PACKH2 | MPY2 | OR | XOR |
| PACKH4 | MPYHI | PACK2 | |
| PACKHL2 | MPYHIR<br>MPYIH<br>MPYIHR | PACKH2 | |
| PACKL4 | MPYIL<br>MPYILR<br>MPYLI | PACKHL2 | |
| PACKLH2 | MPYLIR | PACKLH2 | |
| SHLMB | MPYSU4<br>MPYUS4 | SADD2 | |
| SHRMB | MPYU4 | SADDU4 | |
| SUB2 | MVD | SADDSU2<br>SADDUS2 | |
| SUB4 | ROTL | SHLMB | |
| SUBABS4 | SHFL | SHR2 | |
| SWAP2 | SMPY2 | SHRMB | |
| .L unit | .M unit | .S unit | .D unit |
| SWAP4 | SSHVL | SHRU2 | |
| UNPKHU4 | SSHVR | SPACK2 | |
| UNPKLU4 | XPND2 | SPACKU4 | |
| XOR | XPND4 | SUB2 | |
| | | SWAP2 | |
| | | UNPKHU4 | |
| | | UNPKLU4 | |
| | | XOR | |

**[0043]** Instructions are always fetched eight at a time. This constitutes a fetch packet. The basic format of a fetch packet is shown in Figure 3. The execution grouping of the fetch packet is specified by the p-bit, bit zero, of each

instruction. Fetch packets are 8-word aligned.

**[0044]** The p-bit controls the parallel execution of instructions. The p-bits are scanned from left to right (lower to higher address). If the p-bit of instruction i is 1, then instruction i +1 is to be executed in parallel with (in the same cycle as) instruction i. If the p-bit of instruction i is 0, then instruction i + 1 is executed in the cycle after instruction i. All instructions executing in parallel constitute an execute packet. An execute packet in this embodiment can contain up to eight instructions. All instructions in an execute packet must use a unique functional unit.

**[0045]** The following examples illustrate the conversion of a p-bit sequence into a cycle-by-cycle execution stream of instructions. There are three types of p-bit patterns for fetch packets. These three p-bit patterns result in the following execution sequences for the eight instructions: fully serial; fully parallel; or partially serial. These three sequences of execution are explained more fully below.

**[0046]** The fully serial p-bit pattern depicted in Figure 4A results in this execution sequence is illustrated in Table 5.

Table 5.

| Fully Serial p-bit Pattern Execution Sequence | |
|---|---|
| Cycle | Instructions |
| 1 | A |
| 2 | B |
| 3 | C |
| 4 | D |
| 5 | E |
| 6 | F |
| 7 | G |
| 8 | H |

The eight instructions are executed sequentially.

**[0047]** The fully parallel p-bit pattern depicted in Figure 4B results in this execution sequence is illustrated in Table 6.

Table 6. Fully Parallel p-bit Pattern Execution Sequence

| Cycle | Instructions | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | A | B | C | D | E | F | G | H |

All eight instructions are executed in parallel.

**[0048]** The partially serial p-bit pattern depicted in Figure 4C results in this execution sequence is illustrated in Table 7.

Table 7.

| Partially Serial p-bit Pattern Execution Sequence | | | |
|---|---|---|---|
| Cycle | Instructions | | |
| 1 | A | | |
| 2 | B | | |
| 3 | C | D | E |
| 4 | F | G | H |
| Note that the instructions C, D, and E do not use any of the same functional units, cross paths, or other data path resources. This is also true for instructions F, G, and H. | | | |

**[0049]** The || characters signify that an instruction is to execute in parallel with the previous instruction. In the previous partially serial example, the code would be represented as this:

| | | |
|---|---|---|
| | instruction | A |
| | instruction | B |
| | instruction | C |
| \|\| | instruction | D |
| \|\| | instruction | E |
| | instruction | F |
| \|\| | instruction | G |
| \|\| | instruction | H |

**[0050]** If a branch into the middle of an execution packet occurs, all instructions at lower addresses are ignored. In the partially serial example, if a branch to the address containing instruction D occurs, then only D and E will execute. Even though instruction C is in the same execute packet, it is ignored. Instructions A and B are also ignored because they are in earlier execute packets.

**[0051]** No two instructions within the same execute packet can use the same resources. Also, no two instructions can write to the same register during the same cycle. The following describes each of the resources an instruction can use.

**[0052]** Two instructions using the same functional unit cannot be issued in the same execute packet.

**[0053]** The following execute packet is invalid in the present embodiment:

```
ADD .S1 A0, A1, A2 ; \ .S1 is used for both instructions
|| SHR .S1 A3, 15, A4 ; /
```

**[0054]** The following execute packet is valid in the present embodiment:

```
ADD .L1 A0, A1, A2 ; \ Two different functional units
|| SHR .S1 A3, 15, A4 ; / are used
```

Unprotected pipeline

**[0055]** Since the pipeline of the present embodiment is unprotected, certain code sequences that cause resource conflicts are invalid. An assembler with appropriate code sequence checking capabilities is used to screen out invalid code sequences. Several constraints related to the present embodiment are described below. Other constraints which may apply to the present embodiment are described in US Patent No. 6,112,298 (TI-24946), incorporated herein by reference.

Pipeline Operation

**[0056]** The DSP pipeline has several key features which improve performance, decrease cost, and simplify programming. They are: increased pipelining eliminates traditional architectural bottlenecks in program fetch, data access, and multiply operations; control of the pipeline is simplified by eliminating pipeline interlocks; the pipeline can dispatch eight parallel instructions every cycle; parallel instructions proceed simultaneously through the same pipeline phases; sequential instructions proceed with the same relative pipeline phase difference; and load and store addresses appear on the CPU boundary during the same pipeline phase, eliminating read-after-write memory conflicts.

**[0057]** A multi-stage memory pipeline is present for both data accesses and program fetches. This allows use of high-speed synchronous memories both on-chip and off-chip, and allows infinitely nestable zero-overhead looping with branches in parallel with other instructions.

**[0058]** There are no internal interlocks in the execution cycles of the pipeline, so a new execute packet enters execution every CPU cycle. Therefore, the number of CPU cycles for a particular algorithm with particular input data is fixed. If during program execution, there are no memory stalls, the number of CPU cycles equals the number of clock cycles for a program to execute.

**[0059]** Performance can be inhibited only by stalls from the memory subsystems or interrupts. The reasons for memory stalls are determined by the memory architecture. To fully understand how to optimize a program for speed, the sequence of program fetch, data store, and data load requests the program makes, and how they might stall the CPU should be understood.

**[0060]** The pipeline operation, from a functional point of view, is based on CPU cycles. A CPU cycle is the period during which a particular execute packet is in a particular pipeline stage. CPU cycle boundaries always occur at clock cycle boundaries; however, memory stalls can cause CPU cycles to extend over multiple clock cycles. To understand

the machine state at CPU cycle boundaries, one must be concerned only with the execution phases (E1-E5) of the pipeline. The phases of the pipeline are shown in Figure 11 and described in Table 8.

Table 8.

| Pipeline Phase Description | | | | |
|---|---|---|---|---|
| Pipeline | Pipeline Phase | Symbol | During This Phase | Instruction Types Completed |
| Program Fetch | Program Address Generate | PG | Address of the fetch packet is determined. | |
| | Program Address Send | PS | Address of fetch packet is sent to memory. | |
| | Program Wait | PW | Program memory access is performed. | |
| | Program Data Receive | PR | Fetch packet is expected at CPU boundary. | |
| Program Decode | Dispatch | DP | Next execute packet in fetch packet determined and sent to the appropriate functional units to be decoded. | |
| | Decode | DC | Instructions are decoded at functional units. | |
| Execute | Execute 1 | E1 | For all instruction types, conditions for instructions are evaluated and operands read. Load and store instructions: address generation is computed and address modifications written to register file† Branch instructions: affects branch fetch packet in PG phase† Single-cycle instructions: results are written to a register file† | Single-cycle |
| | Execute 2 | E2 | Load instructions: address is sent to memory† Store instructions and STP: address and data are sent to memory† Single-cycle instructions that saturate results set the SAT bit in the Control Status Register (CSR) if saturation occurs. † Multiply instructions: results are written to a register file† | Stores STP Multiplies |

†This assumes that the conditions for the instructions are evaluated as true. If the condition is evaluated as false, the instruction will not write any results or have any pipeline operation after E1.

Table 8. (continued)

| Pipeline Phase Description | | | | |
|---|---|---|---|---|
| Pipeline | Pipeline Phase | Symbol | During This Phase | Instruction Types Completed |
| | Execute 3 | E3 | Data memory accesses are performed. Any multiply instruction that saturates results sets the SAT bit in the Control Status Register (CSR) if saturation occurs. † | |
| | Execute 4 | E4 | Load instructions: data is brought to CPU boundary† | |
| | Execute 5 | E5 | Load instructions: data is loaded into register† | Loads |

†This assumes that the conditions for the instructions are evaluated as true. If the condition is evaluated as false, the instruction will not write any results or have any pipeline operation after E1.

**[0061]** The pipeline operation of the instructions can be categorized into seven types shown in Table 9. The delay slots for each instruction type are listed in the second column.

Table 9.

| Delay Slot Summary | | |
|---|---|---|
| Instruction Type | Delay Slots | Execute Stages Used |
| Branch (The cycle when the target enters E1) | 5 | E1-branch target EI |
| Load (LD) (Incoming Data) | 4 | E1 - E5 |
| Load (LD) (Address Modification) | 0 | E1 |
| Multiply | 1 | E1- E2 |
| Single-cycle | 0 | E1 |
| Store | 0 | E1 |
| NOP (no execution pipeline-operation) | - | - |
| STP (no CPU internal-results written) | - | - |

**[0062]** The execution of instructions can be defined in terms of delay slots (Table 9). A delay slot is a CPU cycle that occurs after the first execution phase (E1) of an instruction in which results from the instruction are not available. For example, a multiply instruction has 1 delay slot, this means that there is 1 CPU cycle before another instruction can use the results from the multiply instruction.

**[0063]** Single cycle instructions execute during the E1 phase of the pipeline. The operand is read, operation is performed and the results are written to a register all during E1. These instructions have no delay slots.

**[0064]** Multiply instructions complete their operations during the E2 phase of the pipeline. In the E1 phase, the operand is read and the multiply begins. In the E2 phase, the multiply finishes, and the result is written to the destination (dst) register. Multiply instructions have 1 delay slot.

**[0065]** Load instructions have two results: data loaded from memory and address pointer modification.

**[0066]** Data loads complete their operations during the E5 phase of the pipeline. In the E1 phase, the address of the data is computed. In the E2 phase, the data address is sent to data memory. In the E3 phase, a memory read is performed. In the E4 stage, the data is received at the CPU core boundary. Finally, in the E5 phase, the data is loaded into a register. Because data is not written to the register until E5, these instructions have 4 delay slots. Because pointer results are written to the register in E1, there are no delay slots associated with the address modification.

**[0067]** Store instructions complete their operations during the E3 phase of the pipeline. In the E1 phase, the address of the data is computed. In the E2 phase, the data address is sent to data memory. In the E3 phase, a memory write

is performed. The address modification is performed in the E1 stage of the pipeline. Even though stores finish their execution in the E3 phase of the pipeline, they have no delay slots and follow the following rules ( i = cycle):

1) When a load is executed before a store, the old value is loaded and the new value is stored.
   i LDW
   i+1 STW
2) When a store is executed before a load, the new value is stored and the new value is loaded.
   i STW
   i+1 LDW
3) When the instructions are in are in parallel, the old value is loaded and the new value is stored.
   i STW
   i+1 || LDW

**[0068]** Branch instructions execute during the E1 phase of the pipeline five delay slots/CPU cycles after the branch instruction enters an initial E1 phase of the pipeline. Figure 12 shows the branch instruction phases. Figure 13 shows the operation of the pipeline based on clock cycles and fetch packets. In Figure 13, if a branch is in fetch packet n, then the E1 phase of the branch is the PG phase of n+6. In cycle 7 n is in the E1 phase and n+6 is in the PG phase. Because the branch target is in PG on cycle 7, it will not reach E1 until cycle 13. Thus, it appears as if the branch takes six cycles to execute, or has five delay slots.

**[0069]** In Figure 5, fetch packet n, which contains three execute packets, is shown followed by six fetch packets (n+1 through n+6), each with one execution packet (containing 8 parallel instructions). The first fetch packet (n) goes through the program fetch phases during cycles 1-4. During these cycles a program fetch phase is started for each of the following fetch packets.

**[0070]** In cycle 5, the program dispatch (DP) phase, the CPU scans the p-bits and detects that there are three execute packets (k thru k+2) in fetch packet n. This forces the pipeline to stall, which allows the DP phase to start execute packets k+1 and k+2 in cycles 6 and 7. Once execute packet k+2 is ready to move on to the DC phase (cycle 8) the pipeline stall is released.

**[0071]** The fetch packets n+1 through n+4 were all stalled so the CPU would have time to perform the DP phase for each of the three execute packets (k thru k+2) in fetch packet n. Fetch packet n+5 was also stalled in cycles 6 and 7; it was not allowed to enter the PG phase until after the pipeline stall was released in cycle 8. The pipeline will continue as shown with fetch packets n+5 and n+6 until another fetch packet containing multiple execution packets enters the DP phase, or an interrupt occurs.

**[0072]** Figure 6 is a block diagram illustrating a prior art processor 600 that requires execution packets to be aligned within fetch packets. Prior art processor 600 is a VLIW RISC architecture that has an instruction execution pipeline that operates similarly to processor 10 in aspects other than execution packet alignment. Four fetch packets 611-613 are illustrated in Figure 6. These are each fetched sequentially in response to program fetch circuitry in processor 600. Fetch packet 610 comprises an execution packet 620 that has seven useful instructions 620(0)-620(6) that can all be executed in parallel. The next execution packet 621 is in fetch packet 611 and comprises instructions 621(0)-621(4). A third execution packet 622 comprises one useful instruction 622(0). Disadvantageously, a no-operation instruction (NOP) 620(7) must be included in execution packet 620 since execution packets must be aligned within fetch packets. Similarly, NOP instructions 622(1) and 622(2) are placed in execution packet 622 to cause alignment with fetch packet 611. NOP instructions 620(7), 622(1) and 622(2) waste storage resources in processor 600. Note that the last word of each fetch packet has the p-bit set to 0 to indicate the end of an execute packet, such as instruction word 620(7) and 622(2), for example.

**[0073]** Figure 7A is another illustration of fetch packets and execution packets in prior art processor 600 of Figure 6. If an execution packet did not fit evenly within a fetch packet, NOP instructions were inserted in the instruction stream to pad out the fetch packet. For example, in Figure 7A, execution packet E3 cannot fit in the four-word space directly after execution packet E2, therefore four NOP instructions are inserted in the instruction sequence to pad out the fetch packet. Disadvantageously, instruction storage resources are wasted and unexecutable NOP instructions are needlessly fetched.

**[0074]** Figure 7B is an illustration of execution packets spanning fetch packets for the processor of Figure 1. Advantageously, in the present embodiment of processor 10, an execution packet can cross an eight-word fetch packet boundary, thereby eliminating a need to add NOP instructions to pad fetch packets. For example, eight-word execution packet EP1 completely occupies fetch packet 700. Four-word execution packet EP2 partially fills fetch packet 702. Six-word execution packet EP3 does not fit completely within fetch packet 702, however, the first four words EP#(0)-EP3(3) are placed in fetch packet 702 and the last two words EP3(4), EP3(5) are placed in fetch packet 704. Therefore, the last p-bit in a fetch packet is not always set to 0 in processor 10. If the last p-bit of a fetch packet is not zero, then instruction fetch control circuitry in stage 10a (Figure 1) fetches a second fetch packet and extracts instruction words

until a p-bit set to 0 is encountered. This sequence of instruction words is then ordered into a single execution packet, such as execution packet EP3, for example.

**[0075]** Figure 8 is a block diagram of processor 10 of Figure 1, illustrating a sequence of execution packets spanning fetch packets 810-813, according to an aspect of the present invention. For ease of comparison, the same program portion is illustrated here as in Figure 6. For example, execution packet 820 comprises only the seven useful instructions 820(0)-820(6). The last instruction word 820(6) has the p-bit set to 0. Advantageously, the first word 821(0) of execution packet 821 is now placed in fetch packet 810. Note that the p-bit of instruction word 821(0) is set to 1 to indicate that execute packet 821 continues to the next fetch packet 811 where instruction words 821(1)-821(4) are located. Likewise, execution packet 824 spans fetch packets 811 and 812 with instruction word 824(0) located in fetch packet 811 and instruction words 824(1)-824(3) located in fetch packet 812. Execution packet 823 with instruction words 823(0)-823(1) is located within fetch packet 811. Note that the last word of the fetch packet now does not necessarily have the p-bit set to 0, for example instruction words 821(0) and 824(0) both have their p-bit set to one indicating that their associated execution packet spans to the next fetch packet.

**[0076]** Figure 9 is a block diagram of a portion of instruction fetch pipeline stages 10a-10c of processor 10 of Figure 1 that illustrates circuitry for dispatching an execution packet that spans two fetch packets. An eight word fetch packet is received on instruction data bus 941 from an instruction memory/cache into a first latch stage 910. A second latch stage 911 receives the first fetch packet on the next clock cycle while a second fetch packet is received simultaneously into latch stage 910. Multiplexor set 920 has eight individually controlled multiplexors 920(0)-920(7) that each have a first input connected to receive a 32-bit instruction word from first latch stage 910 and a 32-bit instruction word from second latch stage 911. Dispatch control circuitry 921 monitors the p-bit associated with each of the eight instructions in latch stage 910 via p-bit signals 922. An execution packet within latch stage 910 is selected by appropriate controls signals 924 asserted by dispatch control circuitry 921 to multiplexor set 920 and is passed thereby to crosspoint circuitry 930 and on to the various decode circuitry associated with the execution units of processor 10, as illustrated in Figure 8.

**[0077]** If a first execution packet does not entirely fill the first fetch packet, then the second execution packet is selected from second latch stage 911 on a following clock cycle. Dispatch control circuitry 921 monitors second stage p-bit signals 923 and asserts controls signals 924 appropriately to multiplexor set 920 to select the second execution packet from latch stage 911 and it is passed thereby to crosspoint circuitry 930 and on to the various decode/execution units of processor 10.

**[0078]** If the second execute packet spans the first and second fetch packets, then a first portion of the second execute packet is selected from latch stage 911 and a remaining portion is selected from latch stage 910. Dispatch control circuitry 921 monitors p-bit signals 922 and 923 and asserts control signals 924 appropriately so that multiplexor set 920 selects the first portion of the second execute packet from latch stage 911 and the remaining portion from latch stage 910.

**[0079]** Thus, control circuitry 921 is connected to a plurality of instruction positions corresponding to the plurality of instructions of a fetch packet in latch stage 910 and latch stage 911 to determine a boundary of each execute packet. The control circuitry is operable to control selection circuitry 920 in response to an execute packet boundary.

**[0080]** A sequence of instruction fetch packets is fetched, wherein each fetch packet contains a first plurality of instructions. Each fetch packet is examined to determine an execution packet boundary and then a first portion of an execute packet is selected from a first fetch packet and a second portion of a first execute packet from a second fetch packet if the first execute packet boundaries span the first fetch packet and the second fetch packet.

**[0081]** Fetch pipe stalls are generated as needed, as discussed with reference to Figure 5, if latch stage 910 is not free when a fetch packet arrives on instruction bus 941.

**[0082]** Figure 10 is a block diagram of an alternative embodiment of a digital system 1000 with processor core 1001 similar to processor core 10 of Figure 1. A direct mapped program cache 1010b, having 16 kbytes capacity, is controlled by L1 Program (L1P) controller 1010a and connected thereby to the instruction fetch stage 10a. A 2-way set associative data cache 1020b, having a 16 kbyte capacity, is controlled by L1 Data (L1D) controller 1020a and connected thereby to data units D1 and D2. An L2 memory/cache 1030 having four banks of memory, 128 Kbytes total, is connected to L1P 1010a and to L1D 1020a to provide storage for data and programs. External memory interface (EMIF) 1050 provides a 64-bit data path to external memory, not shown, which provides memory data to L2 memory 1030 via extended direct memory access (DMA) controller 1040.

**[0083]** EMIF 1052 provides a 16-bit interface for access to external peripherals, not shown. Expansion bus 1070 provides host and I/O support similarly to host port 60/80 of Figure 1.

**[0084]** Three multi-channel buffered serial ports (McBSP) 1060, 1062, 1064 are connected to DMA controller 1040. A detailed description of a McBSP is provided in U.S. Patent Serial No. 09/055,011 (TI-26204, Seshan, et al) and is incorporated herein reference.

**[0085]** The Very Long Instruction Word (VLIW) CPU of the present invention uses a 256-bit wide instruction to feed up to eight 32-bit instructions to the eight functional units during every clock cycle. The VLIW architecture features controls by which all eight units do not have to be supplied with instructions if they are not ready to execute. The first

bit of every 32-bit instruction determines if the next instruction belongs to the same execute packet as previous instruction, or whether it should be executed in the following clock as a part of the next execute packet. While fetch packets are always 256-bit wide, execute packets can vary in size as shown with reference to Figures 4A-4C. Variable length execute packets are a key memory saving feature distinguishing CPU 1001 from other VLIW architectures.

**[0086]** Advantageously, instruction fetch stage 10a and dispatch stage 10b are constructed according to Figure 9 such that an execution packet can span two fetch packets. Advantageously, NOP instructions are not needed to maintain execution packet alignment in processor 1001.

Other systems

**[0087]** Several example systems which can benefit from aspects of the present invention are described in U.S. Patent 5,072,418, which was incorporated by reference herein, particularly with reference to Figures 2-18 of U.S. Patent 5,072,418. A microprocessor incorporating an aspect of the present invention to improve performance or reduce cost can be used to further improve the systems described in U.S. Patent 5,072,418. Such systems include, but are not limited to, industrial process controls, automotive vehicle systems, motor controls, robotic control systems, satellite telecommunication systems, echo canceling systems, modems, video imaging systems, speech recognition systems, vocoder-modem systems with encryption, and such.

**[0088]** Figure 11 illustrates an exemplary implementation of an example of an integrated circuit 40 that includes digital system 1000 in a mobile telecommunications device, such as a wireless telephone with integrated keyboard 12 and display 14. As shown in Figure 11 digital system 1000 with processor 1001 is connected to the keyboard 12, where appropriate via a keyboard adapter (not shown), to the display 14, where appropriate via a display adapter (not shown) and to radio frequency (RF) circuitry 16. The RF circuitry 16 is connected to an aerial 18. Advantageously, by allowing execute packets to span fetch packets, memory is not wasted with useless NOP instructions. Thus, a smaller memory can be included within the wireless telephone and fewer fetch cycles are required for execution of a given processing algorithm and power consumption is thereby reduced.

Fabrication

**[0089]** Fabrication of digital system 10 or digital system 1000 involves multiple steps of implanting various amounts of impurities into a semiconductor substrate and diffusing the impurities to selected depths within the substrate to form transistor devices. Masks are formed to control the placement of the impurities. Multiple layers of conductive material and insulative material are deposited and etched to interconnect the various devices. These steps are performed in a clean room environment.

**[0090]** A significant portion of the cost of producing the data processing device involves testing. While in wafer form, individual devices are biased to an operational state and probe tested for basic operational functionality. The wafer is then separated into individual dice which may be sold as bare die or packaged. After packaging, finished parts are biased into an operational state and tested for operational functionality.

**[0091]** Thus, a digital system is provided with a processor having an improved instruction fetch and dispatch mechanism. Advantageously, by allowing execute packets to span fetch packets, memory is not wasted with useless NOP instructions.

**[0092]** Advantageously, in some system a smaller memory may result from elimination of NOP instructions.

**[0093]** Advantageously, fewer fetch cycles are required for execution of a given processing algorithm since useless NOP instructions are not fetched.

**[0094]** In another embodiment, fetch packets may be longer or shorter than the present embodiments. For example, a fetch packet may be four instruction words. Likewise, instruction words may be other sizes than 32-bits.

**[0095]** In another embodiment, there may be more or fewer execution units than eight. Likewise, the number of execution units may differ from the size of the fetch packet. For example, a processor may have an eight word fetch packet and have ten execution units, for example. In such an embodiment, an execution packet may exceed the length of a fetch packet and may therefore span more than two fetch packets.

**[0096]** In another embodiment, a different form of dispatching circuitry may be provided to select an execution packet from two or more fetch packets.

**[0097]** As used herein, the terms "applied," "connected," and "connection" mean electrically connected, including where additional elements may be in the electrical connection path. "Associated" means a controlling relationship, such as a memory resource that is controlled by an associated port. The terms assert, assertion, de-assert, de-assertion, negate and negation are used to avoid confusion when dealing with a mixture of active high and active low signals. Assert and assertion are used to indicate that a signal is rendered active, or logically true. De-assert, de-assertion, negate, and negation are used to indicate that a signal is rendered inactive, or logically false.

**[0098]** While the invention has been described with reference to illustrative embodiments, this description is not

intended to be construed in a limiting sense. Various other embodiments of the invention will be apparent to persons skilled in the art upon reference to this description. It is therefore contemplated that the appended claims will cover any such modifications of the embodiments as fall within the true scope of the invention.

## Claims

1. A microprocessor comprising:

   fetch circuitry for fetching instruction fetch packets, wherein each fetch packet contains a first plurality of instructions;
   a plurality of functional units operable to execute a second plurality of instructions in parallel, and
   dispatch circuitry operable to select an execute packet from one or more fetch packets, wherein an execute packet varies in size and contains only a set of instructions that can be executed in parallel on the plurality of functional units, whereby a first execute packet contains a different number of instructions from a second execute packet due to resource constraints.

2. The microprocessor of Claim 1, wherein the first plurality of instructions is equal in number to the second plurality of instructions.

3. The microprocessor of Claim 1 or claim 2, wherein a first execute packet spans a first fetch packet and a second fetch packet.

4. The microprocessor of any one of claims 1 to 3, wherein the dispatch circuitry comprises:

   a first latch to hold the first fetch packet;
   a second latch to hold the second fetch packet;
   selection circuitry to select a first portion of the first execute packet from the first latch and a second portion of the first execute packet from the second latch.

5. The microprocessor of Claim 4, wherein the dispatch circuitry further comprises control circuitry connected to a plurality of instruction positions corresponding to the plurality of instructions of a fetch packet in the first latch and in the second latch to determine a boundary of each execute packet, the control circuitry operable to control the selection circuitry in response to an execute packet boundary.

6. A digital processing system including a microprocessor as claimed in any one of claims 1 to 5.

7. A digital processing system according to Claim 6, which is a cellular telephone, further comprising:

   an integrated keyboard connected to the microprocessor via a keyboard adapter;
   a display, connected to the microprocessor via a display adapter;
   radio frequency (RF) circuitry connected to the microprocessor; and
   an aerial connected to the RF circuitry.

8. A method of operating a microprocessor having a plurality of functional units for executing instructions in parallel, comprising the steps of:

   fetching a sequence of instruction fetch packets, wherein each fetch packet contains a first plurality of instructions;
   examining each fetch packet to determine an execute packet boundary and;
   selecting a first portion of an execute packet from a first fetch packet and a second portion of a first execute packet from a second fetch packet if the first execute packet boundaries span the first fetch packet and the second fetch packet.

9. A method of operating a digital system including a microprocessor having a plurality of functional units for executing instructions in parallel, comprising the steps of:

   fetching a sequence of instruction fetch packets, wherein each fetch packet contains a first plurality of instruc-

tions;
examining each fetch packet to determine an execute packet boundary and;
selecting a first portion of an execute packet from a first fetch packet and a second portion of a first execute packet from a second fetch packet if the first execute packet boundaries span the first fetch packet and the second fetch packet.

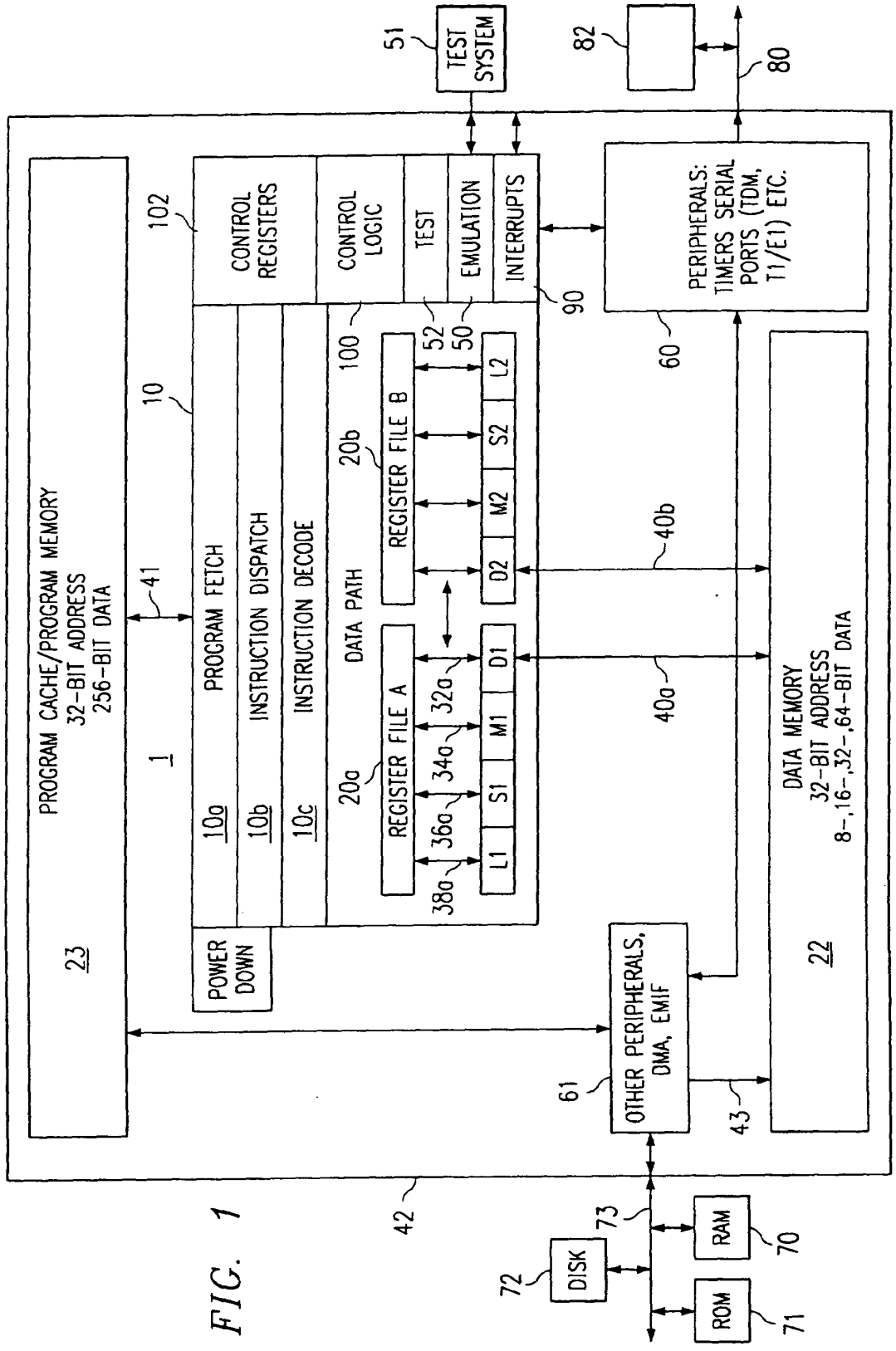

PROGRAM CACHE/PROGRAM MEMORY
32-BIT ADDRESS
256-BIT DATA
23
1
41
10
102
PROGRAM FETCH
INSTRUCTION DISPATCH
INSTRUCTION DECODE
DATA PATH
10a
10b
10c
20a
20b
100
REGISTER FILE A
REGISTER FILE B
L1
S1
M1
D1
D2
M2
S2
L2
32a
34a
36a
38a
40a
40b
CONTROL REGISTERS
CONTROL LOGIC
TEST
EMULATION
INTERRUPTS
52
50
90
POWER DOWN
51
TEST SYSTEM
82
80
PERIPHERALS: TIMERS SERIAL PORTS (TDM, T1/E1) ETC.
60
OTHER PERIPHERALS, DMA, EMIF
61
43
DATA MEMORY
32-BIT ADDRESS
8-,16-,32-,64-BIT DATA
22
42
73
DISK
72
RAM
70
ROM
71

FIG. 1

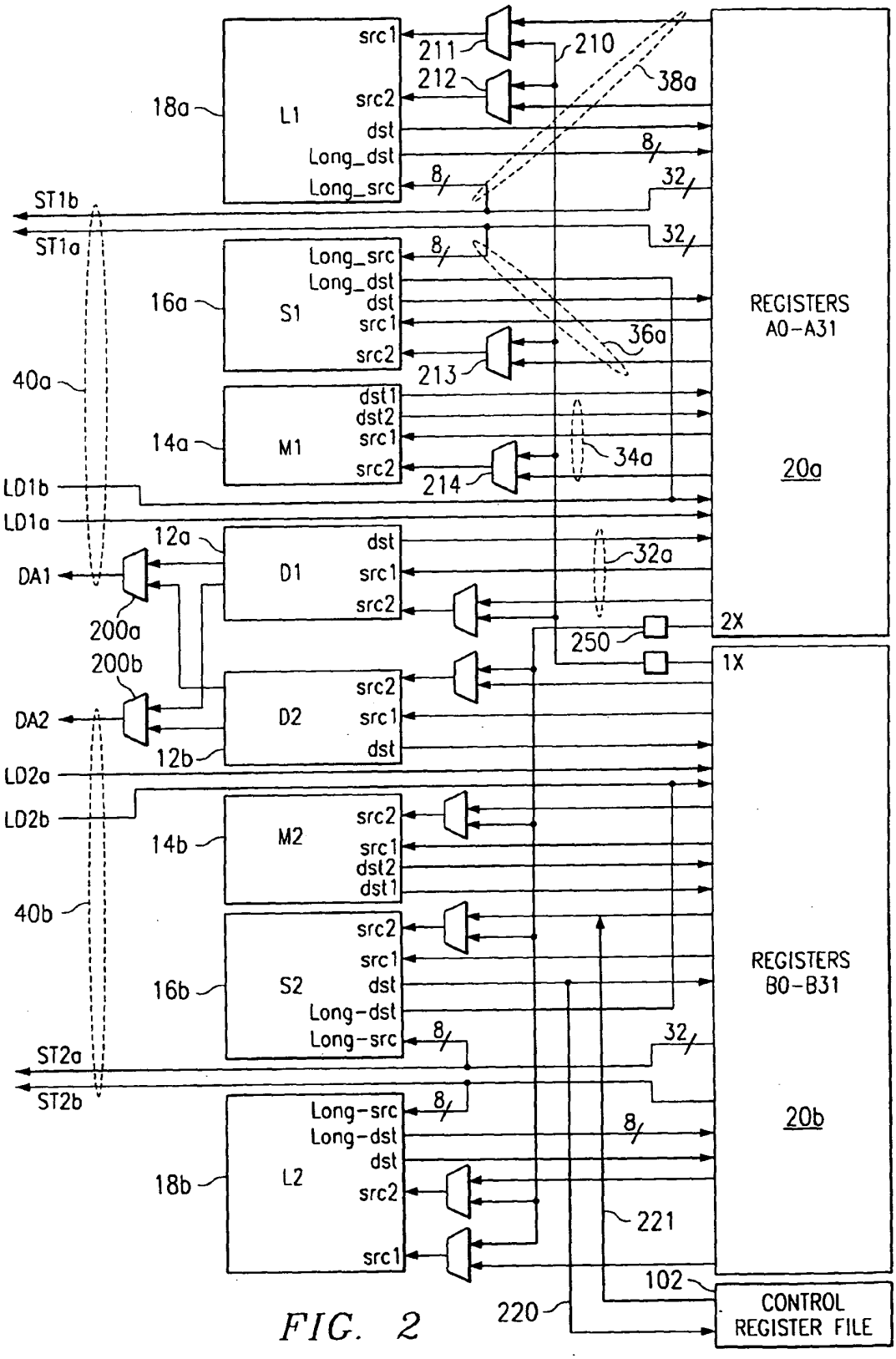

18a
L1
src1
src2
dst
Long_dst
Long_src
211
212
210
38a
8
32
ST1b
ST1a
16a
S1
Long_src
Long_dst
dst
src1
src2
213
36a
40a
14a
M1
dst1
dst2
src1
src2
214
34a
REGISTERS
A0-A31
20a
LD1b
LD1a
12a
D1
dst
src1
src2
32a
DA1
200a
200b
250
2X
1X
D2
DA2
12b
LD2a
LD2b
14b
M2
16b
S2
Long-dst
Long-src
40b
ST2a
ST2b
18b
L2
221
102
CONTROL
REGISTER FILE
220
REGISTERS
B0-B31
20b

*FIG. 2*

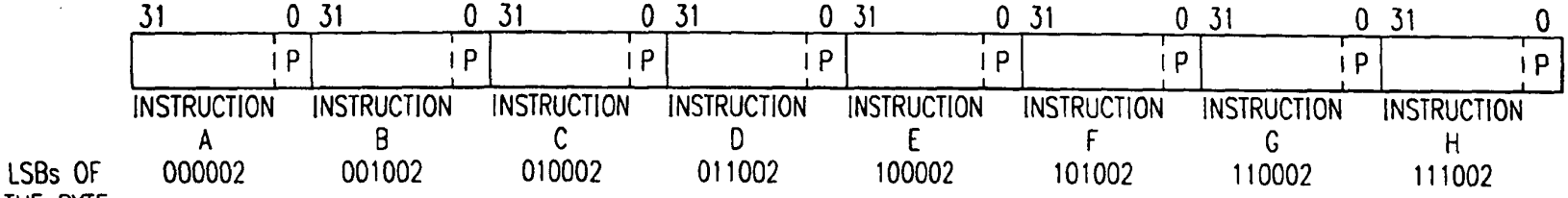



*FIG. 3*

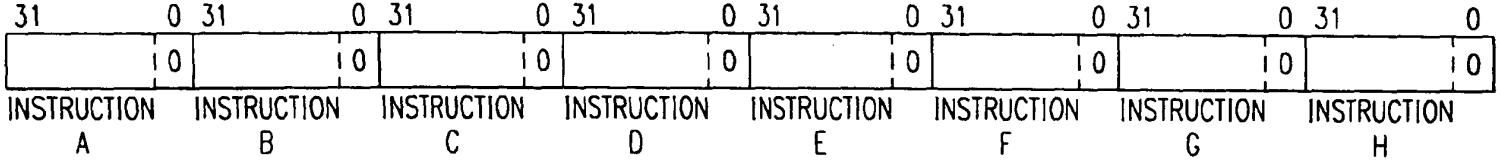



*FIG. 4A*

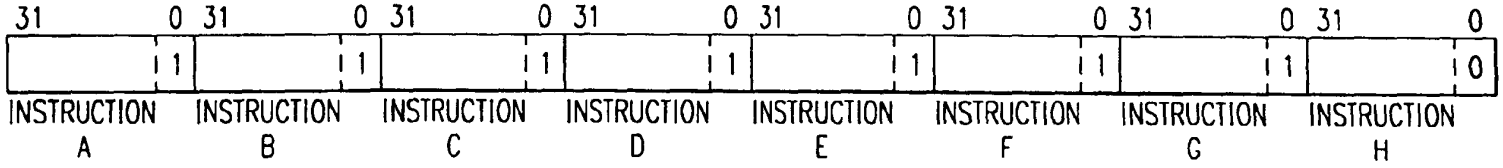



*FIG. 4B*

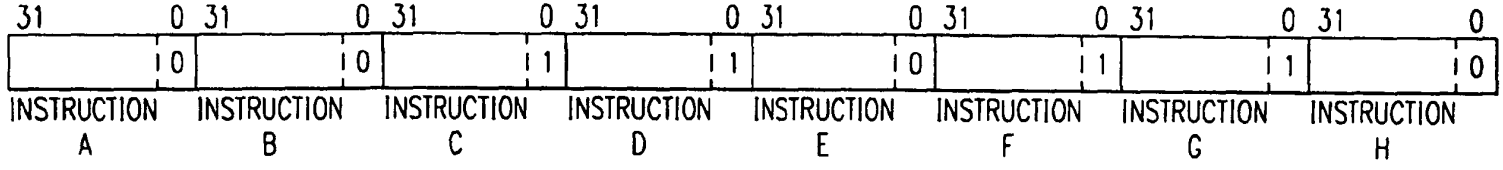



*FIG. 4C*



EP 1 102 165 A1

FIG. 5

| FETCH PACKET | EXECUTE PACKET | CLOCK CYCLE 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| n | k | PG | PS | PW | PR | DP | DC | E1 | E2 | E3 | E4 | E5 | | | | | | | | | | | | |
| n | k+1 | | | | | | DP | DC | E1 | E2 | E3 | E4 | E5 | | | | | | | | | | | |
| n | k+2 | | | | | | | DP | DC | E1 | E2 | E3 | E4 | E5 | | | | | | | | | | |
| n+1 | k+3 | | PG | PS | PW | PR | PR | PR | DP | DC | E1 | E2 | E3 | E4 | E5 | | | | | | | | | |
| n+2 | k+4 | | | PG | PS | PW | PW | PW | PR | DP | DC | E1 | E2 | E3 | E4 | E5 | | | | | | | | |
| n+3 | k+5 | | | | PG | PS | PS | PS | PW | PR | DP | DC | E1 | E2 | E3 | E4 | E5 | | | | | | | |
| n+4 | k+6 | | | | | PG | PG | PG | PS | PW | PR | DP | DC | E1 | E2 | E3 | E4 | E5 | | | | | | |
| n+5 | k+7 | | | | | | | | PG | PS | PW | PR | DP | DC | E1 | E2 | E3 | E4 | E5 | | | | | |
| n+6 | k+8 | | | | | | | | | PG | PS | PW | PR | DP | DC | E1 | E2 | E3 | E4 | E5 | | | | |

FIG. 7A
(PRIOR ART)

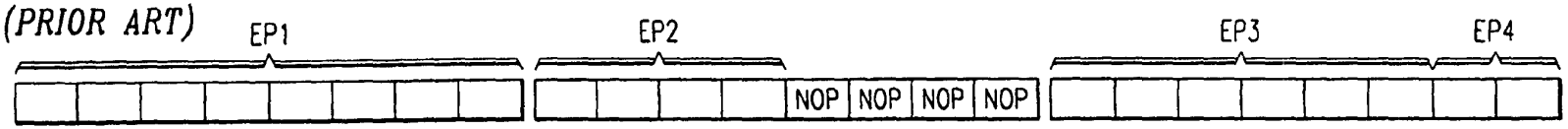
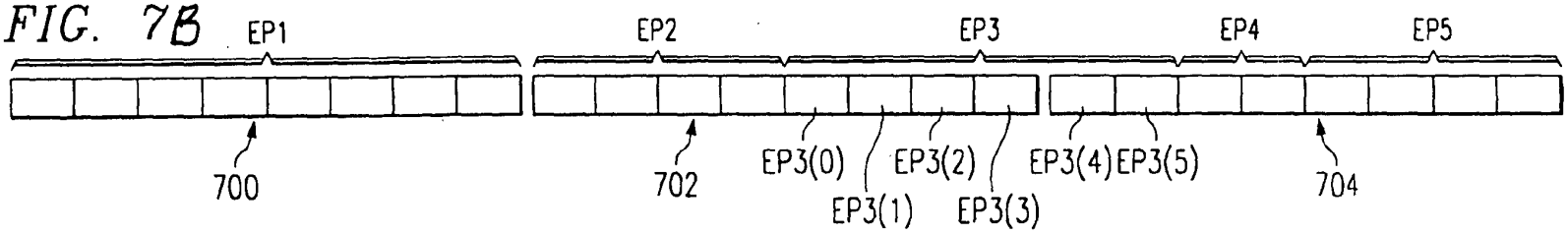



FIG. 7B



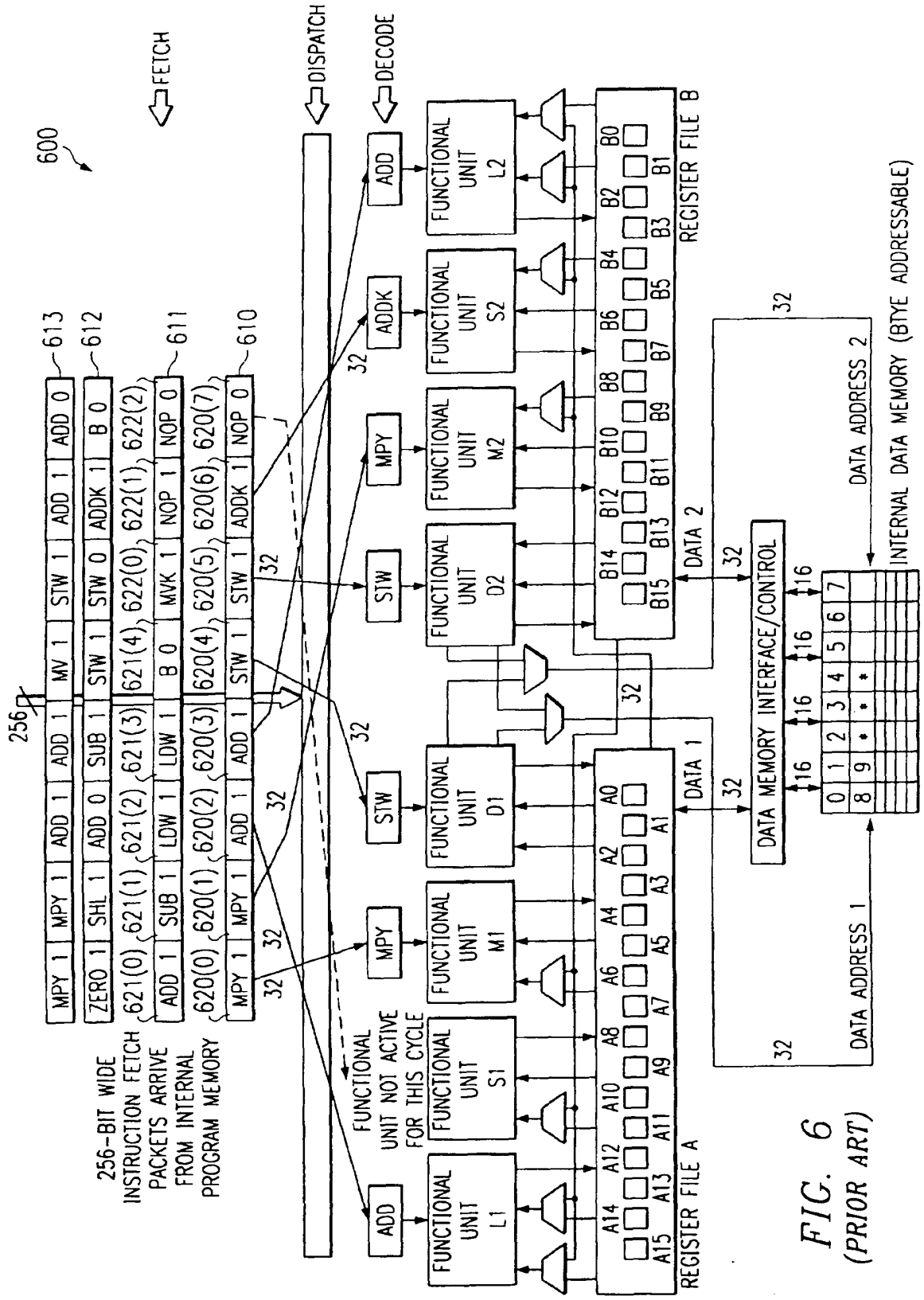



*FIG. 6*
*(PRIOR ART)*

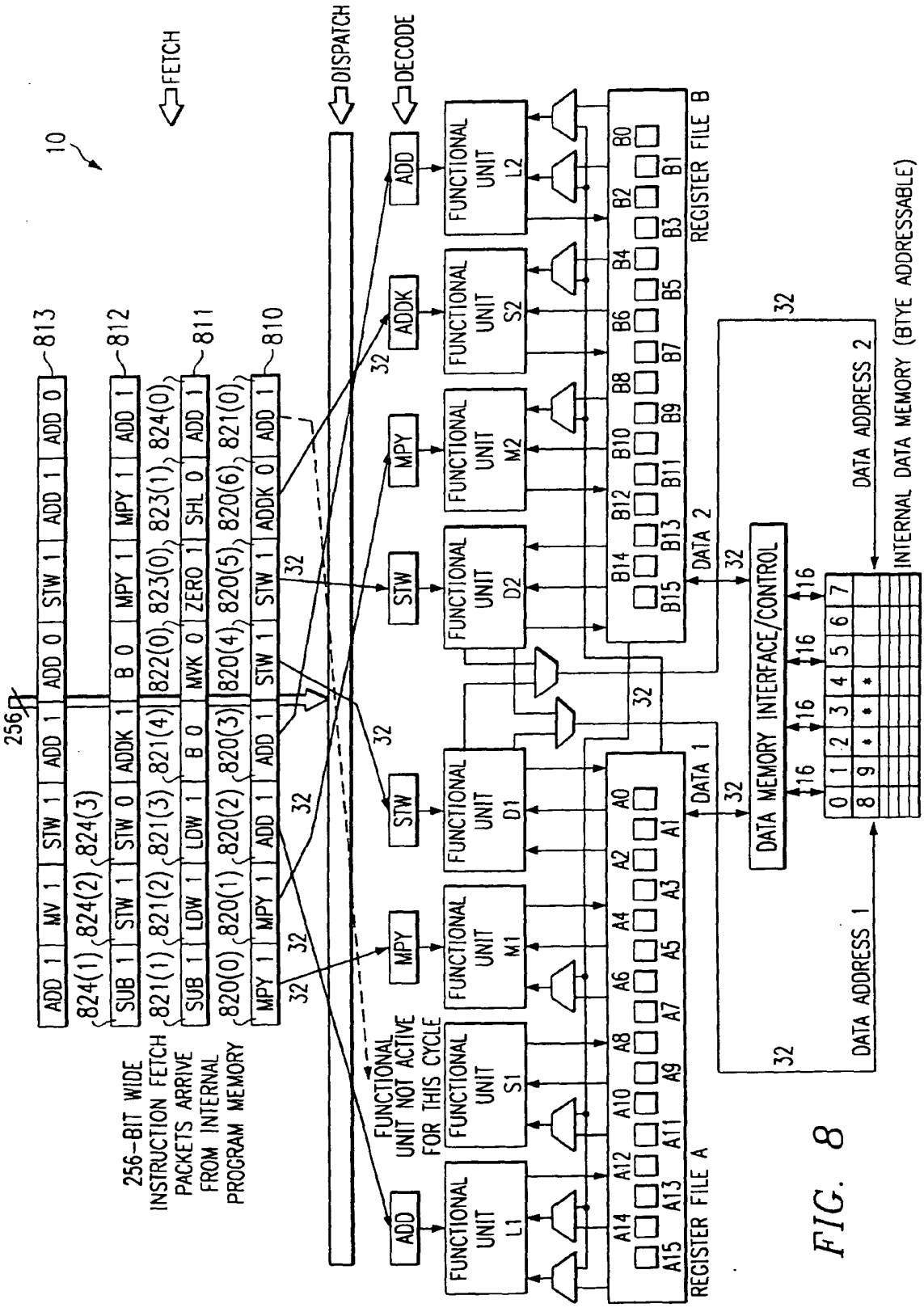

10
FETCH
DISPATCH
DECODE
256
813
812
811
810
ADD 1 | MV 1 | STW 1 | ADD 1 | ADD 0 | STW 1 | ADD 1 | ADD 0
824(1) 824(2) 824(3)
SUB 1 | STW 1 | STW 0 | ADDK 1 | B 0 | MPY 1 | MPY 1 | ADD 1
821(1) 821(2) 821(3) 821(4) 822(0) 823(0) 823(1) 824(0)
SUB 1 | LDW 1 | LDW 1 | B 0 | MVK 0 | ZERO 1 | SHL 0 | ADD 1
820(0) 820(1) 820(2) 820(3) 820(4) 820(5) 820(6) 821(0)
MPY 1 | MPY 1 | ADD 1 | ADD 1 | STW 1 | STW 1 | ADDK 0 | ADD 1
256-BIT WIDE INSTRUCTION FETCH PACKETS ARRIVE FROM INTERNAL PROGRAM MEMORY
FUNCTIONAL UNIT NOT ACTIVE FOR THIS CYCLE
ADD
MPY
STW
ADDK
FUNCTIONAL UNIT L1
FUNCTIONAL UNIT S1
FUNCTIONAL UNIT M1
FUNCTIONAL UNIT D1
FUNCTIONAL UNIT D2
FUNCTIONAL UNIT M2
FUNCTIONAL UNIT S2
FUNCTIONAL UNIT L2
REGISTER FILE A
REGISTER FILE B
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15
DATA 1
DATA 2
32
DATA MEMORY INTERFACE/CONTROL
16
DATA ADDRESS 1
DATA ADDRESS 2
INTERNAL DATA MEMORY (BTYE ADDRESSABLE)
0 1 2 3 4 5 6 7
8 9 * * * * * *

FIG. 8

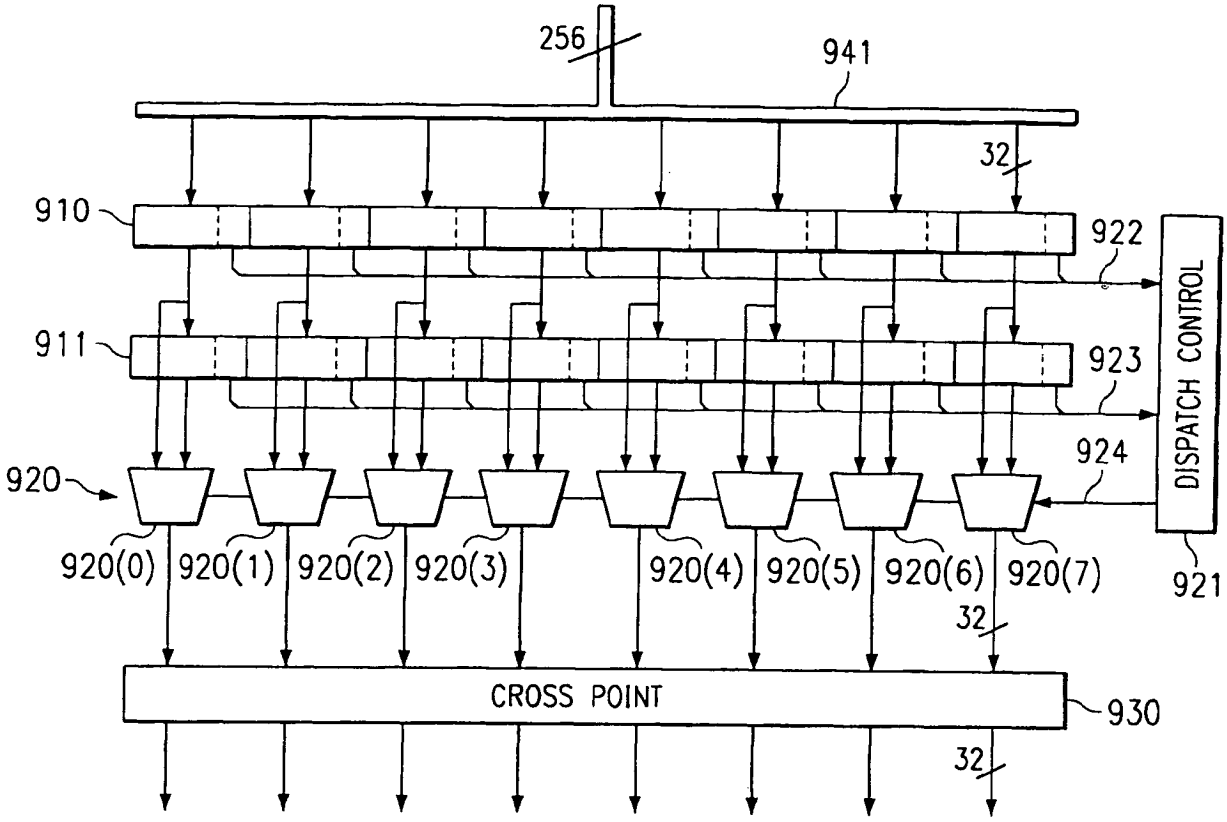

256
941
32
910
922
911
923
DISPATCH CONTROL
920
924
920(0)
920(1)
920(2)
920(3)
920(4)
920(5)
920(6)
920(7)
921
32
CROSS POINT
930
32

FIG. 9

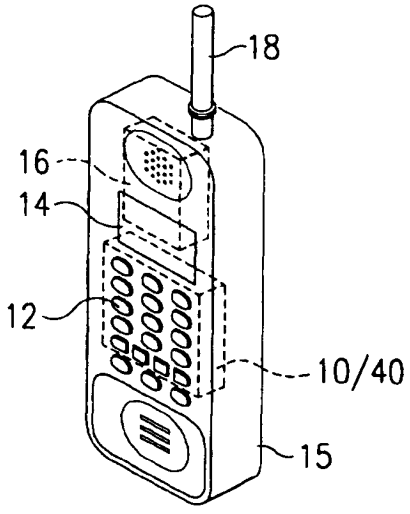

18
16
14
12
10/40
15

FIG. 11

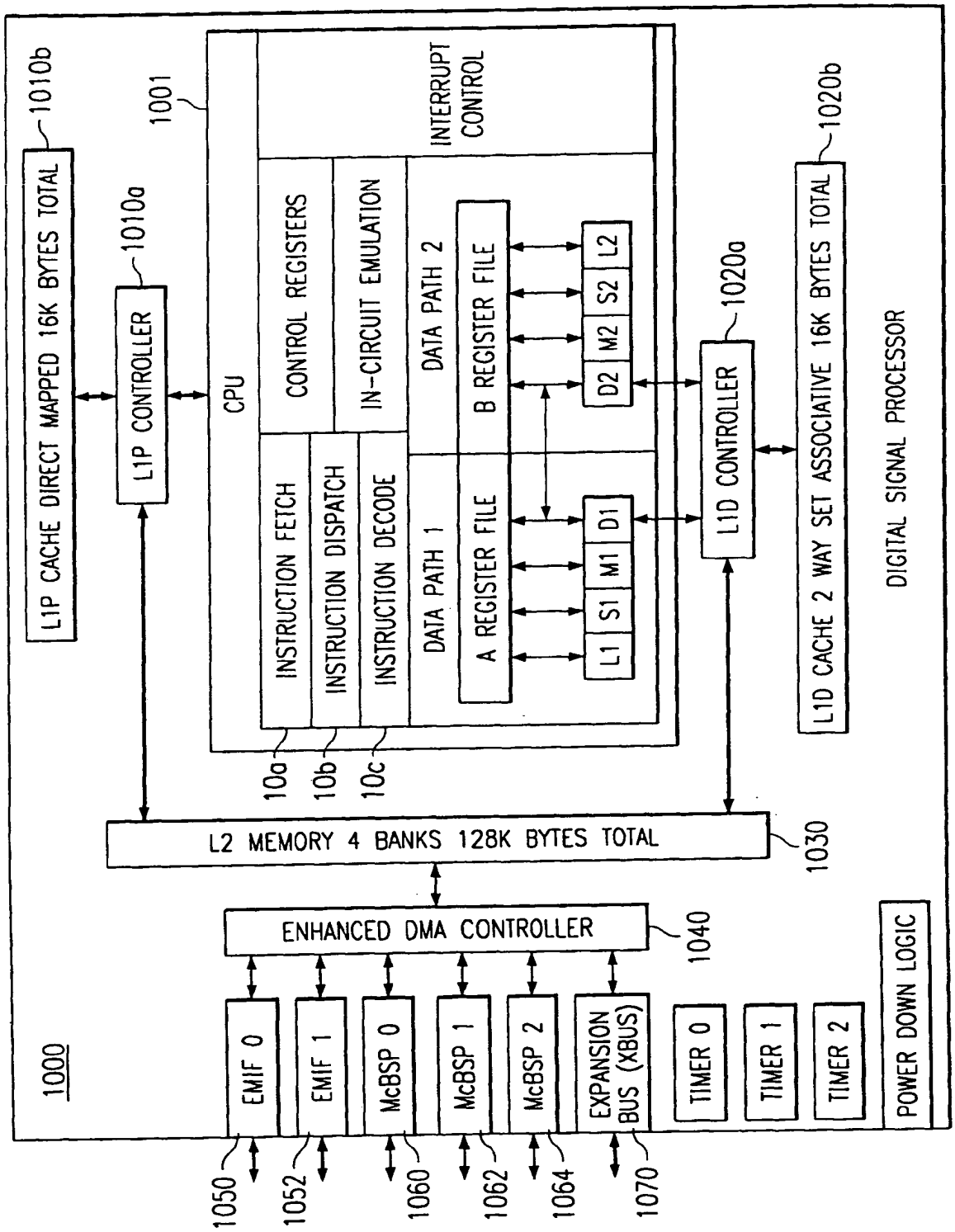

1000
L1P CACHE DIRECT MAPPED 16K BYTES TOTAL
1010b
L1P CONTROLLER
1010a
1001
CPU
INSTRUCTION FETCH
INSTRUCTION DISPATCH
INSTRUCTION DECODE
CONTROL REGISTERS
IN-CIRCUIT EMULATION
INTERRUPT CONTROL
DATA PATH 1
A REGISTER FILE
L1 S1 M1 D1
DATA PATH 2
B REGISTER FILE
D2 M2 S2 L2
10a
10b
10c
L1D CONTROLLER
1020a
L1D CACHE 2 WAY SET ASSOCIATIVE 16K BYTES TOTAL
1020b
DIGITAL SIGNAL PROCESSOR
L2 MEMORY 4 BANKS 128K BYTES TOTAL
1030
ENHANCED DMA CONTROLLER
1040
EMIF 0
EMIF 1
McBSP 0
McBSP 1
McBSP 2
EXPANSION BUS (XBUS)
TIMER 0
TIMER 1
TIMER 2
POWER DOWN LOGIC
1050
1052
1060
1062
1064
1070

FIG. 10

European Patent Office

# EUROPEAN SEARCH REPORT

Application Number
EP 00 31 0114

## DOCUMENTS CONSIDERED TO BE RELEVANT

| Category | Citation of document with indication, where appropriate, of relevant passages | Relevant to claim | CLASSIFICATION OF THE APPLICATION (Int.Cl.7) |
|---|---|---|---|
| X | EP 0 667 571 A (HEWLETT PACKARD CO)<br>16 August 1995 (1995-08-16)<br>* page 3, line 49 - page 4, line 28 *<br>* figure 3 * | 1-3,5,6,<br>8,9 | G06F9/38 |
| P,X | WO 99 63417 A (ERICSSON TELEFON AB L M)<br>9 December 1999 (1999-12-09)<br>* abstract *<br>* page 10, line 9 - line 24 *<br>* figure 3 * | 1,3,4,6,<br>8,9 | |
| A | US 5 459 844 A (BLANER BARTHOLOMEW ET AL)<br>17 October 1995 (1995-10-17)<br>* column 15, line 6 - line 32 *<br>* figure 11 * | 1,8,9 | |
| A | EP 0 855 648 A (TEXAS INSTRUMENTS INC)<br>29 July 1998 (1998-07-29)<br>* page 5, line 43 - page 6, line 7 * | 1,8,9 | |

TECHNICAL FIELDS SEARCHED (Int.Cl.7)

G06F

The present search report has been drawn up for all claims

| Place of search | Date of completion of the search | Examiner |
|---|---|---|
| THE HAGUE | 5 March 2001 | Moraiti, M |

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

## ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 00 31 0114

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

05-03-2001

| Patent document cited in search report | Publication date | Patent family member(s) | Publication date |
|---|---|---|---|
| EP 0667571 A | 16-08-1995 | US 5974534 A | 26-10-1999 |
| | | DE 69510966 D | 02-09-1999 |
| | | DE 69510966 T | 09-12-1999 |
| | | JP 7253887 A | 03-10-1995 |
| WO 9963417 A | 09-12-1999 | AU 4661499 A | 20-12-1999 |
| US 5459844 A | 17-10-1995 | BR 9102128 A | 24-12-1991 |
| | | CA 2038264 C | 27-06-1995 |
| | | CZ 9100934 A | 12-07-1995 |
| | | EP 0463296 A | 02-01-1992 |
| | | HU 57920 A | 30-12-1991 |
| | | JP 2046576 C | 25-04-1996 |
| | | JP 7078738 B | 23-08-1995 |
| | | PL 165585 B | 31-01-1995 |
| | | SK 93491 A | 13-09-1995 |
| | | RU 2109333 C | 20-04-1998 |
| | | US 5355460 A | 11-10-1994 |
| EP 0855648 A | 29-07-1998 | JP 10232779 A | 02-09-1998 |

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82